NEASIM: A General-Purpose Computer Simulation Program for Load-Loss Analysis of Multistage Central Office Switching Networks

By R. F. GRANTGES and N. R. SINOWITZ

(Manuscript received November 7, 1963)

Blocking probability is the most frequently required performance characteristic in traffic studies of complex central office switching networks. Determining this quantity without actual measurement is a difficult task. To aid the communications system designer, a simulation program has been prepared which produces useful estimates of blocking probability for a large class of networks. The program is based on a simplified mathematical model for the analysis of switching networks developed by C. Y. Lee, and thus differs from conventional simulators in that it simulates a mathematical model rather than a traffic-handling system.

Although Lee's model is widely used, its utility has been limited by computational difficulties encountered in networks of realistic size and complexity. This limitation is in most practical cases removed by the program, which features rapid input preparation, short computer runs, and specification of the desired precision of the results as an input parameter. Moreover, the program allows for the incorporation of more a priori *information about the actual behaviour of switching networks than is included in Lee's model, thereby leading to a more accurate estimate of blocking probability.*

The simulator has been programmed for the IBM 7090 computer, but the concepts are machine independent.

CONTENTS

I. INTRODUCTION

Determining the traffic performance of complex multistage central office switching systems without actual measurement can be a major problem for the communications engineer. While probability theory has been successfully applied to a wide variety of telephone traffic problems,[1,2,3] a precise formulation of a mathematical model completely describing the multistage switching system has thus far not been found.

No systematic approach exists which completely accounts for the gross complexity encountered in large-scale congestion systems, but several authors have contributed significantly to the theory, notably C. Jacobaeus,[4,6] K. Lundkvist,[5] A. Jensen,[7] C. Y. Lee,[8] A. Elldin,[9] R. Fortet,[10] and P. LeGall.[11] More recently, V. E. Beneš[12-15] has initiated "an attempt to describe a comprehensive point of view towards the subject of connecting systems."[12] Although the engineer does not yet have a comprehensive theory, he does have a valuable tool in computer simulation.

Simulation of telephone traffic flow has a long history in the Bell System. As early as 1907, a rudimentary simulation was undertaken to improve switchboard performance. Artificial traffic was generated by a card-drawing technique, and the simulation was used to verify a semi-mathematical analysis of the loads which could be handled by a team of operators meeting an average delay criterion. In the ensuing years, simulation techniques have been aids in the study of complex traffic problems, such as the effect of limited sources on graded multiple capacities, the efficiency of random slipped multiples, the capacities of various alternate routing plans, and the distribution of delays under various trunking plans. The traffic load capacity of the No. 1 crossbar network was largely determined by the load-loss relationships in the link and junctor patterns obtained from elaborate simulations begun in 1936. This was the first time that the capacity of a largely complete system had come under study by simulation methods. A 10,000-line No. 5 crossbar office was simulated in 1948 by a specially designed machine

which coordinated the efforts of four operators, providing significant data for the traffic engineering of this system.[16]

In recent years the high-speed electronic digital computer has proved an effective tool in large-scale traffic simulations.[17-22] For this class of simulations, special computer programs are written which usually contain:

(*i*) a logical description of the system under consideration,

(*ii*) a procedure for generating and offering traffic to the system, and

(*iii*) a method for extracting and recording the desired system characteristics.

These programs may be called "special-purpose" simulators — in the sense that they are written for the purpose of studying a specific traffic-handling system. A variety of performance data may be obtained, including:

(*i*) probability of blocking at various loads (load-loss data);

(*ii*) delay distribution, including average delay on calls delayed; and

(*iii*) mean queue lengths.

Of these, the most frequently required datum is the probability of loss (or delay).

These simulation programs have produced a large amount of useful information, but their application has not been widespread because of the considerable programming effort required. To reduce programming effort, various "general-purpose" traffic simulation programs have been written.[24,25,26] However, it must be understood that each program is only "general" with respect to a particular class of traffic systems.

The multistage central office switching system is an example of a class of traffic-handling systems for which no general-purpose simulator has heretofore been written, although much has been accomplished by special-purpose simulations written for specific switching network arrangements.[27] Because the use of these network simulation programs has been greatly restricted by the cumbersome programming and input preparation required, a strong need has developed for a quick, easy-to-use, general-purpose simulation technique.

To meet this need, the authors have developed a computer program which, with a minimum of user effort, will produce useful estimates of blocking probability for a very large class of multistage switching networks. The program is based on a simplified mathematical model of switching networks developed by C. Y. Lee,[8] and differs in approach from programs referred to earlier in that a complete description of the *traffic-handling system itself* is not given to the computer; rather, the program simulates the behavior of Lee's *analytical model*. Deriving its

name from this view of its operation, the program has been acronymously named NEASIM — NEtwork Analytical SIMulator.

While Lee's model is probably the most widely used analytical model for multistage matching networks, its utility has been severely limited by the computational difficulties associated with networks of realistic size and complexity. This limitation is in most practical cases removed by the program, which features rapid input preparation and short (hence economical) computer runs. Furthermore, unlike many simulations in which the reliability of results must be assessed on an *a posteriori* basis, the analytical simulator admits of an *a priori* appraisal so that desired precision becomes an input parameter.

The probability linear-graph model, basic to the NEASIM approach, is described in Section II, where its use in switching network analysis is explained. The philosophy of the program together with a general description is presented in Section III. Two key program routines are described in Section IV. Reliability considerations are given in Section V. Finally, Section VI discusses program modifications which can increase the validity of the NEASIM estimate.

II. THE PROBABILITY LINEAR-GRAPH MODEL

In 1955 C. Y. Lee,[8] extending the earlier work of Kittredge and Molina, presented a simplified mathematical model for the analysis of switching networks. Since the NEASIM program simulates the behavior of this model, a description of the model is given (Section 2.1). An example to illustrate how the model is applied is given in Section 2.2; the computational difficulties which may be encountered in the analysis of practical networks are then discussed — thus pointing up the need for the simulation program. Section 2.3 explores further the notion of blocking probability for a network and the use of the linear-graph model in determining this quantity.

2.1 *The Model*

Consider a crosspoint network in which each input can be connected to any output by the operation of appropriate crosspoints. Let $P_t(j,k)$ be the probability that all paths through the network between input j and output k are busy at time t.

Associate with each link l_i of the network a binary-valued random variable $X_t^{(i)}$ whose value represents the state of the link at time t. The NEASIM convention is: 0 represents idle and 1 represents busy.

Since the concern of telephone traffic engineers is with "busy-hour" traffic, it is usually assumed that

(a) *the busy-idle distributions of the link random variables are stationary (or homogeneous) in time.*

That is, for any N,

$$\Pr\{X_{t_n}^{(i)} = \delta_n \; ; \; n = 1, \cdots, N\} = \Pr\{X_{t_n+h}^{(i)} = \delta_n \; ; \; n = 1, \cdots, N\}$$

for all h, where $\delta_n = 0$ or 1 and the index i runs over all the links in the network. A consequence of this assumption is that $P_t(j,k)$ is also stationary in time, for all j and k, and the subscript t will henceforth be omitted.

Let us now fix our attention on two terminals, one on each side of a crosspoint network in which

(b) *all of the switches are nonblocking,**

and in which

(c) *there is no connection path between any switch and itself.*

Then the configuration of possible paths through the network between the two terminals can be represented by a two-terminal cycle-free linear graph with directed branches, in which the nodes of the graph represent network switches and the directed branches of the graph represent network links. Consider, for example, the network depicted in Fig. 1. The possible path configuration seen between terminals A and B is indicated by heavy lines, and the corresponding graph is as shown in Fig. 2(a).

Next, assume that

(d) *$P(j,k)$ is independent of j and k,*

and we can speak of $P(j,k) = B$ as the probability of blocking of the network. The notion of blocking probability will be further explored in Section 2.3.

Finally, Lee makes the simplifying assumption:

(e) *the link random variables, $X^{(i)}$, are independent.*

This assumption, which is frequently made to render analysis manageable, is the principal weakness of the model and will cause the results to depart from reality in varying degrees — depending on the particular network. In general the model will tend to overestimate blocking. The problem of obtaining realistic results is discussed further in Section VI.

Each of a large class of switching networks can therefore be repre-

---

* Lee's requirement that the switches be nonblocking is actually not restrictive, and can be relaxed by an appropriate adjustment of the link occupancies.

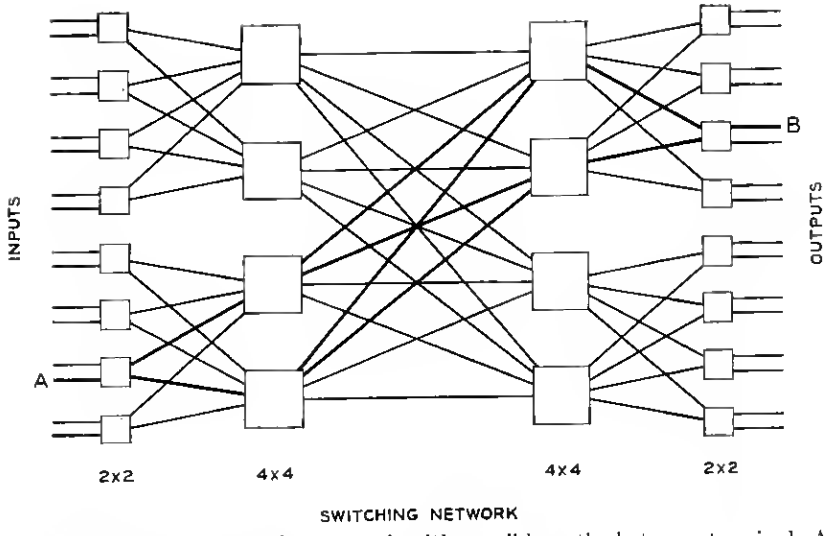

Fig. 1 — Simple crosspoint network with possible paths between terminals A and B indicated by heavy lines.

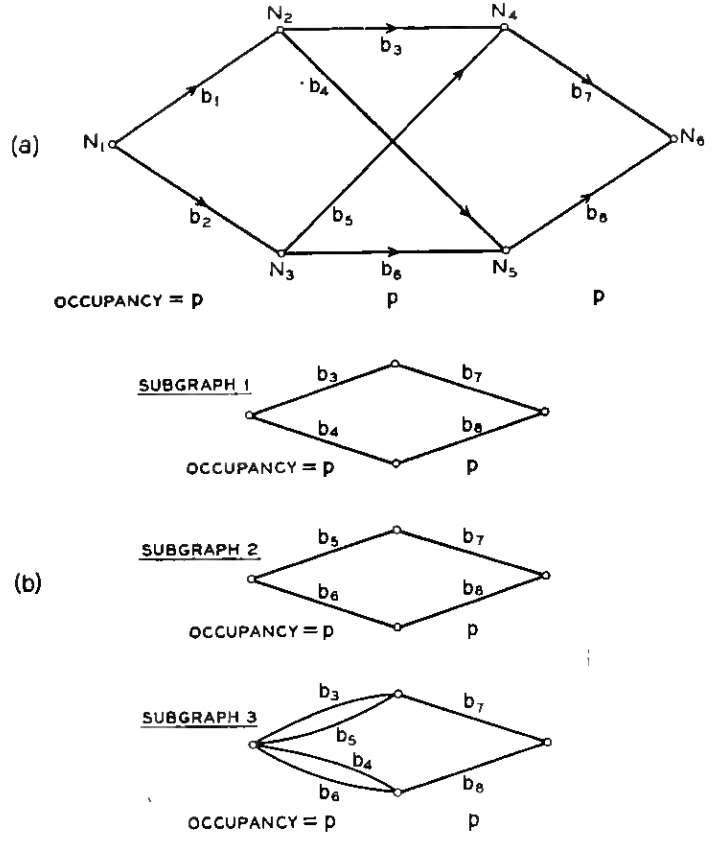

Fig. 2 — (a) GRAPH of the network of Fig. 1, with occupancy p on each of the branches. (b) The subgraphs of the GRAPH.

sented by a simplified model called, by Lee, a *two-terminal probability linear-graph* in which

(*i*) switches are represented by nodes, and links by directed branches, and

(*ii*) assumptions (a)–(c) hold.

The mathematical object, the two-terminal probability linear-graph, will be referred to as GRAPH in the sequel.

Once the GRAPH of a network is obtained, the calculation of the blocking probability can proceed in a straightforward manner.

2.2 *An Example*

As a first illustration, consider the GRAPH of Fig. 2(a). Let E be the event that there is a path through the GRAPH,* E_i be the event that branch b_i is idle, and $p_i = \Pr\{b_i \text{ is busy}\}$. Then

$$E = A_1 \cup A_2 \cup A_3 \cup A_4$$

where the paths, A_i, are

$$A_1 = E_1 \cap E_3 \cap E_7$$

$$A_2 = E_1 \cap E_4 \cap E_8$$

$$A_3 = E_2 \cap E_5 \cap E_7$$

$$A_4 = E_2 \cap E_6 \cap E_8 .$$

The blocking probability is then

$$\begin{aligned} B &= 1 - \Pr\{E\} \\ &= 1 - \Pr\{A_1 \cup A_2 \cup A_3 \cup A_4\} \\ &= 1 - \sum_{i=1}^{4} \Pr\{A_i\} + \sum_{\substack{i,j=1 \\ i<j}}^{4} \Pr\{A_i \cap A_j\} \\ &\quad - \sum_{\substack{i,j,k=1 \\ i<j<k}}^{4} \Pr\{A_i \cap A_j \cap A_k\} + \Pr\{A_1 \cap A_2 \cap A_3 \cap A_4\}. \end{aligned}$$

Now the assumption of independence gives

$$\begin{aligned} \Pr\{E_i \cdots E_j\} &= \Pr\{E_i\} \cdots \Pr\{E_j\} \\ &= q_i \cdots q_j \qquad q_i = 1 - p_i \end{aligned}$$

* Having adopted the GRAPH model, we speak of a "path through the GRAPH" rather than "a path through the network," and say that "a branch is busy or idle" rather than "a link is busy or idle."

whence, for the case in which $p_i = p$ for all i,

$$B = q^8 - 4q^7 + 2q^6 + 4q^5 - 4q^3 + 1. \tag{1}$$

In general, given a GRAPH G with m different link occupancies $p_1, \cdots, p_m$, the procedure just illustrated will yield the *blocking polynomial* of the GRAPH:

$$B_G = B_G(p_1, \cdots, p_m).$$

As a computational tool the utility of the GRAPH model decreases with increasing complexity of the GRAPH's geometrical structure. When the GRAPH geometry grows more complex, the blocking polynomial B_G becomes cumbersome — admitting a greater possibility for error in its determination. Moreover, once B_G is found, one still has to substitute numerical values for $p_1, \cdots, p_m$ into the polynomial to obtain a result. As an example, for the GRAPH of moderate complexity shown in Fig. 3, the blocking polynomial is

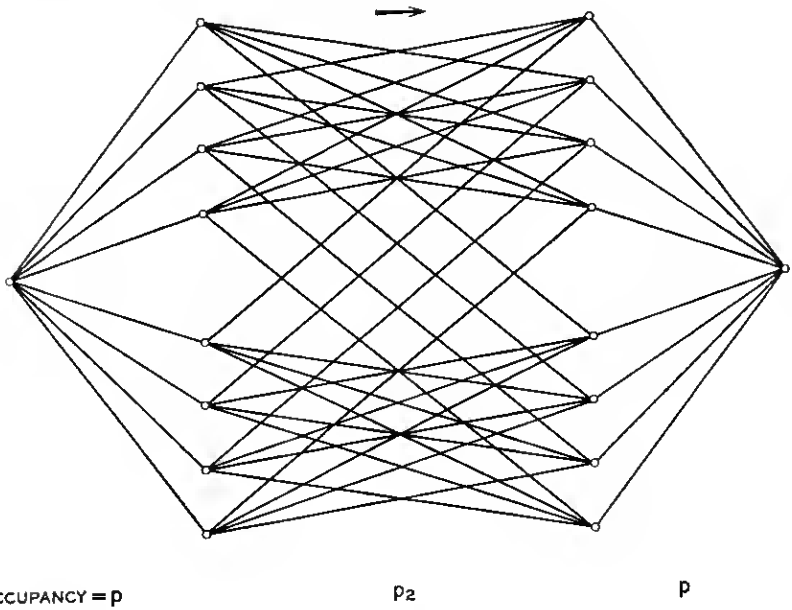

Fig. 3 — A GRAPH of moderate complexity.

$$
\begin{aligned}
B = {} & p^8 \\
& + p^7q(8A^4) \\
& + p^6q^2(4A^8 + 24A^4B^2) \\
& + p^5q^3(24A^4B^4 + 32A^3B^3C) \\
& + p^4q^4(8A^4C^4 + 48A^2B^4C^2 + 8B^6D + 6B^8) \\
& + p^3q^5(32AB^3C^3D + 24B^4C^4) \\
& + p^2q^6(4C^8 + 24B^2C^4D^2) \\
& + pq^7(8C^4D^4) \\
& + q^8(D^8)
\end{aligned}
\tag{2}
$$

where:

$$
\begin{aligned}
A &= p + qp_2 \\
B &= p + qp_2^2 \\
C &= p + qp_2^3 \\
D &= p + qp_2^4 \\
q &= 1 - p.
\end{aligned}
$$

Faced with computing B in (2) over a range of occupancies, an engineer would surely resort to a computer. The essential computational difficulty is that one is confronted with expressions of the form

$$\Pr\{A_i \cup \cdots \cup A_j\}$$

where the paths $A_i, \cdots, A_j$ are not disjoint.

It is interesting to note that the method of approach just described — which may be called the "path enumeration approach" — is not the only way to proceed and is indeed not the most efficient. A second procedure for finding the blocking polynomial may be called the "combinatorial approach" and is best illustrated by example.

Since all the branches in the GRAPH of Fig. 2(a) are busy with probability p, a moment's reflection shows that the blocking probability can be written as

$$B = p^2 + qp \cdot B_{\text{subgraph 1}} + pq \cdot B_{\text{subgraph 2}} + q^2 \cdot B_{\text{subgraph 3}}$$

where $B_{\text{subgraph 1}}$, $B_{\text{subgraph 2}}$, $B_{\text{subgraph 3}}$ are, respectively, the blocking probabilities of the GRAPHs indicated in Fig. 2(b). But by inspection we have

$$B_{\text{subgraph 1}} = (1 - q^2)^2$$
$$B_{\text{subgraph 2}} = (1 - q^2)^2$$
$$B_{\text{subgraph 3}} = [1 - (1 - p^2)q]^2,$$

so that

$$B = p^2 + 2pq(1 - q^2)^2 + q^2[1 - (1 - p^2)q]^2$$

is the blocking polynomial (1) expressed in another form.

The GRAPH of Fig. 4 is more representative of the type of GRAPH

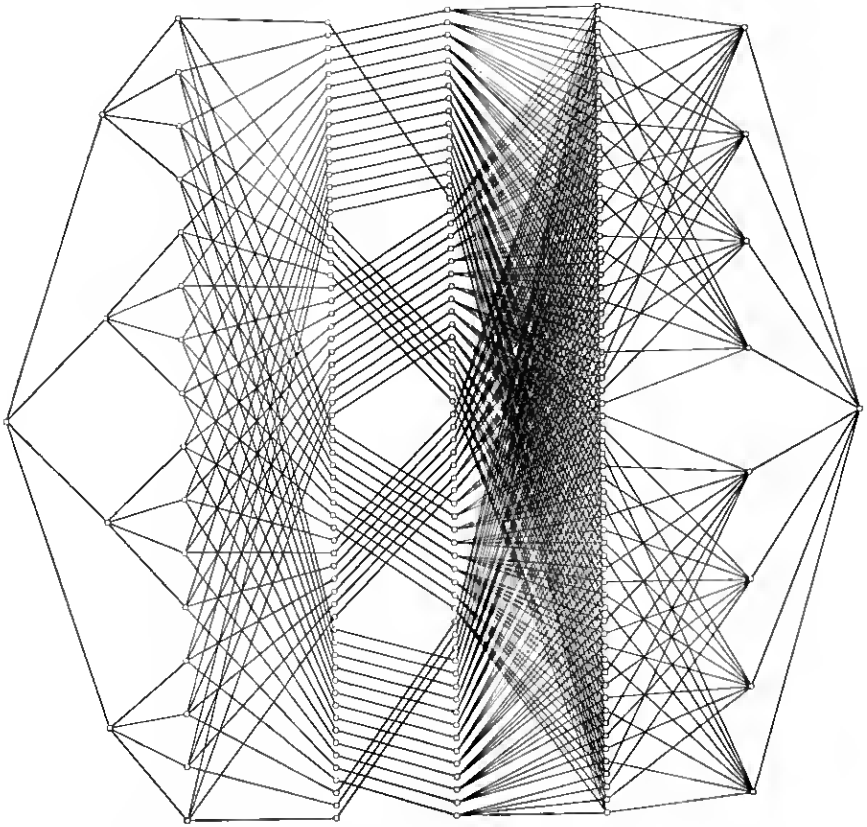

Fig. 4 — Typical GRAPH geometry of realistic central office network.

encountered in modern central office networks. Determination of the blocking polynomial in this case — by either the path enumeration or combinatorial approaches — is a formidable task indeed. If the GRAPH model is to be useful to the modern engineer, some means of handling such complex GRAPHs must be made available.

When seeking performance measures of a network, the engineer is not really concerned with the explicit polynomial representation $B_G(p_1, \cdots, p_m)$; what he would like is a curve (or set of curves) displaying this functional relationship. The simulation program described in the following sections, when given the link occupancies $p_1, \cdots, p_m$, produces, with predictable precision, a numerical estimate of B_G.

2.3 *Blocking Probability and the Linear-Graph Model*

When several kinds of traffic are handled with different disciplines in a network which may have several characteristic graphs, blocking probability for the network can only be meaningfully defined relative to the persons or terminals encountering blocking. For example, the blocking encountered by a call originating and terminating in the same central office may not be the same as the blocking encountered by an incoming call. In general, a network is required to handle several "classes" of connections and the engineer is concerned with the blocking probability for each of these classes. We shall limit our discussion of blocking probability to one class, that is, a subset of all input-output pairs in which each input-output pair has the same graph and for which it is reasonable to suppose that the traffic between every input-output pair is identical* with that of every other pair. Without loss of generality we can therefore assume that the network has one class, so that when speaking of the blocking probability of the network, we shall mean the blocking probability of the class. (These remarks form the basis of assumption (d) of Section 2.1.)

Having thus limited ourselves to one class of connections, we have still to define the blocking probability of a network in a manner which is in agreement with the generally familiar definitions. Here we again encounter difficulty. The authors agree wholeheartedly with Beneš (Ref. 15, p. 2805), that "In fact, not even the definition (let alone the calculation) of the probability of blocking has received adequate treatment . . . ."

Syski (Ref. 3, p. 198), after a long series of prefatory remarks, defines two quantities, time congestion $S(t)$ and call congestion $\pi(t)$ for the case

* That is, every input calls every output at the same rate with the same holding time distribution and with the same lost calls disposition.

of a simple full-access trunk group. The time congestion is the probability that all trunks in the group are busy at time t; the call congestion is the conditional probability that the group is blocked when a call arrives at the instant t. Under the input assumptions and for equilibrium conditions, time and call congestions are independent of time and are denoted by S and π, respectively. S is equivalent to the fraction of time during which congestion is encountered, while π is equivalent to the fraction of calls encountering congestion, and the two quantities will, in general, differ.

Of these, of course, the measure of most concern to the network designer is call congestion. Now if it is assumed that calls originate completely independent of the state of the network, time congestion will equal call congestion. Such an assumption is unjustified if calls cannot originate from busy lines, since time congestion conventionally includes busy line periods while call congestion excludes them. It is, however, reasonable to assume that idle pairs of terminals originate calls at a constant rate independent of the state of the network. In particular, there must be no change in the calling rate after a blocked call. If, under this assumption, time congestion is modified to include only periods in which both lines are idle, it will be equal to call congestion. Actually, even if the foregoing assumption is not met, the time between calls is likely to be much longer than the time taken by the network to return to equilibrium, so that, again, the modified time congestion will be close to the call congestion.

With a suitable choice of branch occupancies, Lee's model allows the computation of call congestion. Alternatively, the branch occupancies may be chosen so as not to reflect the requirement that only idle terminals are to be considered, thus allowing the computation of time congestion. In either case the computed results will be subject to the error introduced by inaccurate assumptions in the model.

Before viewing the probability linear-graph model in the light of the above remarks, it is well to make an observation on the underlying philosophy of the model. Let us perform the following conceptual experiment in a real network under a particular set of (equilibrium) traffic conditions. Suppose that we fix our attention on a particular representative input-output pair (j,k) and examine closely that portion of the network seen between them, i.e., their graph. It is reasonable to believe that, were we to examine the detailed traffic pattern within the graph for a sufficiently long time, we would ultimately come to have complete knowledge of the busy-idle state behavior of the graph links under the particular traffic conditions. The experiment could be repeated under other equilibrium traffic conditions, so that we would eventually be able

to describe completely the behavior of the graph links under all equilibrium conditions. Assuming that the particular input-output pair and connection graph studied are truly representative of the entire network (or at least of an entire connection class), we could then "discard" the rest of the network and determine the blocking probability of the network under various equilibrium conditions by computations or simulations based only on the connection graph and our complete knowledge of its behavior.

That such complete knowledge could be obtained is a practical impossibility. In the absence of complete knowledge, assumptions can be and, indeed, must be made about the detailed behavior of the graph based on such *a priori* knowledge as we have. It is reasonable to suppose that, as the assumptions made approach the real behavior of the graph, the blocking probability determined from the graph will approach the real blocking of the network. A belief in the fundamental soundness of this reasoning constitutes the basic philosophy of the graph model approach to the determination of blocking probability, beginning with Molina and continuing with C. Y. Lee and the NEASIM program.

The assumption made by Lee of link independence [(e) of Section 2.1], while obviously omitting much *a priori* knowledge of graph behavior, possesses the practical advantage of allowing computation of the blocking polynomial where graph geometry does not prohibit. The basic NEASIM program allows evaluation of the polynomial for all graphs meeting Lee's restrictions.

The utility of the results obtainable from Lee's model is well known. When the specific branch occupancies are chosen rationally,* the calculated blocking agrees well enough with real blocking figures (obtained from full-scale simulation or measurement) for many engineering and design purposes. Accuracy can be improved by "calibrating" Lee's results against real values where they are available. Furthermore, computed values lacking in absolute accuracy will reveal relative differences between networks and between various traffic conditions in the same network.

The NEASIM program, to which the remainder of this paper is devoted, is basically designed to estimate the value of the blocking polynomial. This portion of its design and use is described in Sections III–V. The design also allows additional assumptions regarding the detailed traffic behavior of connection graph link states to be incorporated in the graph model. When more *a priori* information is included,

* That is, chosen to reflect the requirement that the input-output terminals j,k are idle by (usually) subtracting the load contributed by the terminals j,k from the assumed carried link loads.

the validity of the simulation results improves as anticipated. This aspect of the NEASIM program is described in Section VI.

III. THE NEASIM PROGRAM

This part of the paper is devoted to a presentation of the program. The basic viewpoint or philosophy of the NEASIM approach is given in Section 3.1. A general description of the program is contained in Section 3.2.

3.1 *Philosophy of the Program*

We saw in the preceding section how the GRAPH model provides — in theory at least — a method for the analysis of a large class of switching networks and how the model is impractical as a tool for networks with complex geometrical structure. To evolve a practical tool, we shall change our method of approach.

Suppose we are given a GRAPH G with branch occupancies

$$p_1 , \cdots , p_m .$$

Until now we were concerned with the explicit blocking polynomial $B_G = B_G(p_1 , \cdots , p_m)$. However, we can think of $B_G(p_1 , \cdots , p_m)$ as a curve in Euclidean $(m + 1)$-space, and set as our goal a precise approximation to this curve. This point of view immediately suggests the use of simulation techniques.

For example, if all the branches in the GRAPH of Fig. 2(a) are busy with probability p, then $B_G = B_G(p)$ is a curve in 2-space. If a computer simulation were to be performed to give an approximation to $B_G(p)$, then we would require a computer program containing an algorithm which, when repeated n times, will produce an estimate $B_G^{(n)}(p)$ such that

$$\lim_{n \to \infty} B_G^{(n)}(p) = B_G(p)$$

and for which we have confidence limits on the absolute error,

$$| B_G^{(n)}(p) - B_G(p) | ,$$

for all n.

Consider the eight branches of the GRAPH of Fig. 2(a). Although all of them are busy with probability p, at any one instant each of the branches is either busy or idle — and for this particular configuration of busy and idle branches there is or is not a path through the graph. Thus, in each repetition of the above mentioned algorithm,

(*i*) we assign busy-idle states to each of the eight branches in such a way that probability of busy in each case is p;

(*ii*) after the assignment has been made we determine whether or not there is a path through the graph for the given assignment.

Let $B_G^{(n)}(p)$ represent the proportion of n repetitions of the algorithm when no path through the graph was found. We would then expect that $\lim_{n\to\infty} B_G^{(n)}(p) = B_G(p)$; that is, we would expect our estimator to converge to the true blocking curve. In general, if the GRAPH G has m different occupancies $p_1, \cdots, p_m$, the program should produce an estimator $B_G^{(n)}(p_1, \cdots, p_m)$ converging to $B_G(p_1, \cdots, p_m)$.

The preceding heuristic remarks were intended to outline the essential approach taken by the NEASIM program. The remainder of Section III is devoted to a description of the program itself. Sections IV and V contain the arguments which show that the estimator indeed converges to the true blocking curve, and how confidence statements about the precision of the results are obtained.

3.2 *Program Description*

The following four sections describe the NEASIM program. Section 3.2.1 takes a "macroscopic" point of view, beginning with an account of the input and then proceeding to give a broad outline of the program. Section 3.2.2 sketches the layout of data in the computer memory. Section 3.2.3 discusses the organization and operation of the NEASIM algorithm. The salient features of the program flow are shown in Fig. 5. Storage requirements and execution speed are given in Section 3.2.4.

3.2.1 *Macroscopic Description*

NEASIM was written for the IBM 7090 computer. The input consists of punched cards which we categorize as Graph Definition Cards and Simulation Definition Cards. Since the notion of a probability-linear graph implies a geometrical configuration together with an occupancy assignment on the branches, the computer must be supplied with both these types of information. The geometrical configuration is read into the computer via the Graph Definition Cards and the various occupancies are read in via the Simulation Definition Cards.

Graph Definition Cards

The information punched on these cards includes

(1) the total number of nodes,

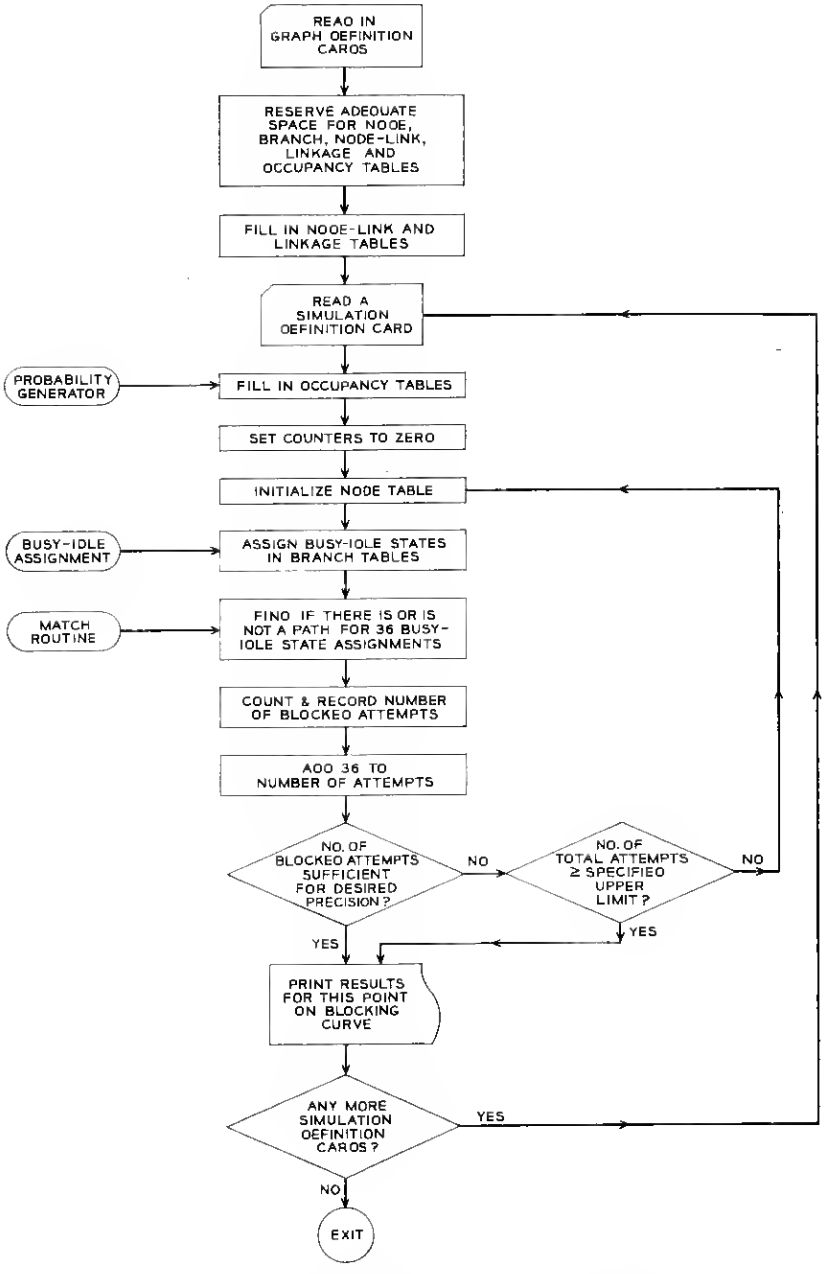

Fig. 5 — Flow diagram for the NEASIM program.

(2) the total number of branches,
(3) the total number of branch occupancies,
(4) the occupancy associated with each of the branches, and
(5) the interconnection scheme between the various nodes.

Simulation Definition Cards

Suppose there are m different branch occupancies for the GRAPH in question. These m values are punched on a Simulation Definition Card. For every set of values of the occupancies $p_1, \cdots, p_m$ — that is, for every point on the blocking curve $B_G(p_1, \cdots, p_m)$ — there is one Simulation Definition Card.

The estimator $B_G^{(n)}(p_1, \cdots, p_m)$ will converge to $B_G(p_1, \cdots, p_m)$ as $n \to \infty$. But the computer must be instructed as to when to terminate the run. Now, the NEASIM process is such that it is possible to request that the error $| B_G^{(n)} - B_G |$ lie within a given percentage of the true value B_G. This "desired precision" information is also punched on the Simulation Definition Card. It may happen, however (as will be the case whenever B_G is very small) that, in order to obtain the requested percentage error, the total number of repetitions of the NEASIM algorithm will perforce be exceedingly large. Since a lengthy computer run is economically undesirable, and since a high degree of precision is usually not needed when the blocking probability is so very low, an upper limit on the number of repetitions, n, is also supplied to the computer by being punched on the Simulation Definition Card. If, after a run has been made, a greater degree of precision is still needed, it is possible to "pick up where we left off" and continue the simulation in another computer run.

The Program

The Graph Definition Cards are read into the computer first. With this information the program constructs, in effect, a map in the computer memory of the geometrical configuration of the GRAPH. Moreover, the program associates with each branch an occupancy p_i — *whose value is as yet unspecified.* The first Simulation Definition Card is then read in, and the program *now* assigns the appropriate values to $p_1, \cdots, p_m$. Once this information is obtained, the program is ready to execute the NEASIM algorithm, which consists essentially of two parts:

(*i*) a busy-idle assignment is made on all of the branches in accordance with the specified occupancies $p_1, \cdots, p_m$;

(*ii*) the presence or absence of a path is determined for the particular assignment.

These two steps are repeated again and again until such time as the estimate $B_G^{(n)}(p_1, \cdots, p_m)$ has been found.

The results for this point on the blocking curve are printed out, and the second Simulation Definition Card (if there is one) is read. The program then goes through the preceding steps for this second set of values of $p_1, \cdots, p_m$ until the estimate for this point on the blocking curve has been obtained. When all the Simulation Definition Cards have been processed, the program run ends.

3.2.2 *Memory Organization*

After the Graph Definition Cards have been read in, the program prepares five main tables as follows:

(1) Node table
(2) Node-Link table
(3) Branch tables
(4) Occupancy tables
(5) Linkage table.

The Node Table

The size (i.e., the number of words) of the Node table is equal to the number of nodes in the GRAPH, there being a one-to-one correspondence between the words in this table and the nodes in the GRAPH. These words are used by the program to indicate, after a particular iteration of the NEASIM algorithm, whether or not there is a path from each particular node to the first node.

The Node-Link Table

The size of the Node-Link table is also equal to the total number of nodes in the GRAPH, with each word corresponding to a particular GRAPH node. For each node the table indicates

(*i*) the number of branches leaving the node — connecting to nodes more distant from the origin, and

(*ii*) a reference to a section of the Linkage table where further information on each branch is stored.

The Node-Link and Linkage tables together constitute the program's map of the GRAPH geometry. The other tables provide storage for busy-idle indications and path information.

The Branch Tables

There are as many Branch tables as there are different occupancies in the GRAPH. For occupancies $p_1, \cdots, p_m$ there will be m Branch tables, which we denote by $BRT_{(1)}, \cdots, BRT_{(m)}$. The size of $BRT_{(i)}$ equals the number of branches in the GRAPH which are busy with probability p_i. There is a one-to-one correspondence between a particular word in $BRT_{(i)}$ and a particular branch in the GRAPH. The association between the words in the Branch tables and the particular GRAPH branches is part of the information stored in the Linkage table. The Branch table words are used by the program to store the busy-idle state of every GRAPH branch on each iteration of the NEASIM algorithm. The total storage required for the Branch tables equals the number of branches in the GRAPH.

The Occupancy Tables

There are m Occupancy tables, $OCT_{(1)}, \cdots, OCT_{(m)}$, corresponding respectively to $BRT_{(1)}, \cdots, BRT_{(m)}$. The size of $OCT_{(i)}$ is an input parameter of the program chosen to be large compared with $BRT_{(i)}$. Every bit in the table $OCT_{(i)}$ contains a binary one with probability p_i. The Occupancy tables are used by the program to supply random busy-idle states for assignment to the branches of the GRAPH on each iteration of the NEASIM algorithm.

The Linkage Table

As previously mentioned, this table contains the detailed interconnection information between the various nodes in the GRAPH. The size of the table is equal to the number of branches in the GRAPH. Each word in the table represents a branch, say b_j, in the GRAPH and contains

(*i*) the address of a word in the Node table which corresponds to the node to which b_j leads, and

(*ii*) the address of a word in a Branch table which stores the current busy-idle state of the branch b_j.

Consider, for example, the GRAPH of Fig. 2(a) and suppose that the occupancy of branches b_1 and b_2 is p_1; the occupancy of branches b_3, b_4, b_5 and b_6 is p_2; and the occupancy of branches b_7 and b_8 is p_3. The tables which the program would prepare are indicated in Fig. 6. The symbolic addresses, such as NW1, BRW3, etc., have been chosen for illustrative purposes only.

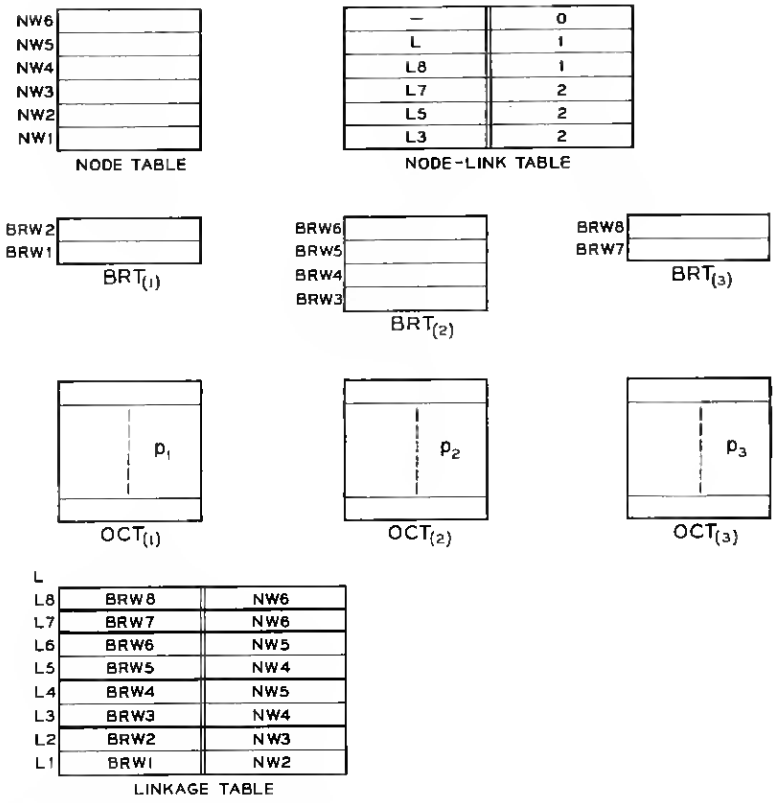

Fig. 6 — Computer memory layout for the GRAPH of Fig. 2 with occupancies p_1, p_2 and p_3.

3.2.3 *Organization of the NEASIM Algorithm*

In a previous section it was mentioned that the NEASIM algorithm requires

(*i*) an assignment of busy-idle states on the branches, and

(*ii*) a method of searching for the existence of a path.

These tasks are performed by three program segments which we shall call

(*i*) the PROBABILITY GENERATOR,

(*ii*) the BUSY-IDLE ASSIGNMENT, and

(*iii*) the MATCH routine.

For the sake of program efficiency (speed) the task of assigning busy-idle states to the branches is divided into two parts. The function of the PROBABILITY GENERATOR is to generate tables of busy-idle

bits — the Occupancy tables. The BUSY-IDLE ASSIGNMENT routine assigns busy-idle states to branches on each iteration of the algorithm. The function of the MATCH routine is to find whether or not there is a path, given a particular assignment of busy-idle states on the branches. Discussion of the internal logic of these programs will be deferred to Section IV. For the present we will be concerned only with their functions.

After the Graph Definition Cards have been read, enough information is available for the program to reserve appropriate space for the Node, Node-Link, Branch, Occupancy and Linkage tables. Once space allotment has been made and the necessary entries filled into the Node-Link and Linkage tables, the program reads the first Simulation Definition Card. Among other parameters, this card specifies the m different branch occupancies desired. At this point the PROBABILITY GENERATOR fills in the Occupancy tables. When this routine has completed its task, each bit in $\mathrm{OCT}_{(i)}$, $i = 1, \cdots, m$, will contain a binary one with probability p_i and a zero with probability $1 - p_i$. (Recall that the NEASIM convention is that one represents busy and zero represents idle.) The contents of the Occupancy tables remain unaltered for the entire time that the program is seeking an estimate for one point on the blocking curve.

The storage of many computers is organized into sequentially numbered words, each of which consists of a fixed number of contiguous bits, and each of which is addressable by the stored program for logical and algebraic manipulation. The number of bits in an IBM 7090 word is 36, so that each word in the Occupancy tables represents 36 independent busy-idle states — thus allowing for 36 independent repetitions of the NEASIM algorithm.

After the Occupancy tables have been prepared, various counters are set to zero and the Node table is initialized. As the contents of the Node table indicate for each node the presence or absence of a path from the particular node to the origin, the program takes the attitude of the man from Missouri and assumes there is no path until one is proven to exist. Thus the words of the Node table are initially set to all 1's except for the first node, which always contains zeros.

At this point the program enters the BUSY-IDLE ASSIGNMENT routine. To assign busy-idle states to the branches, this routine steps through each of the Branch tables and for each word in $\mathrm{BRT}_{(i)}$, a word from $\mathrm{OCT}_{(i)}$ is selected at random and its contents duplicated in the Branch table word. When BUSY-IDLE ASSIGNMENT has been completed, the program enters the MATCH routine.

Using the linkage information stored in the Node-Link and Linkage tables, the MATCH routine performs its logic on the Branch and Node

tables to find whether or not there is a path through the graph for each of the 36 independent busy-idle configurations. The operation of this routine is analyzed in Section 4.2.

When MATCH has completed its work, the number of I's in the word of the Node table corresponding to the terminal node in the GRAPH (NW6 in Fig. 6) will be the number of times there was no path through the graph in the 36 trials. This number of blocked attempts is noted and 36 is entered into a "run-length" counter. The Node table is reinitialized, busy-idle states reassigned and the algorithm repeated again and again. The repetitions will terminate when one of two situations occurs at the end of the MATCH routine. Either

(*i*) enough attempts have been scored to guarantee the precision called for on the Simulation Definition Card, or

(*ii*) the maximum number of attempts specified on the card has been exceeded.

When the repetitions terminate, the proportion of blocked attempts is calculated and the results are printed out. The next Simulation Definition Card is read and the process starts over for the next point on the blocking curve.

3.2.4 *Storage Requirements and Execution Speed*

The largest GRAPH that can be handled by the present version of NEASIM is determined by the core storage available in a particular computer. Total storage required is given by the expression

$$P + 2(N + B) + O$$

where:

P is the number of storage locations required by the NEASIM program itself (about 2000 words),

N is the number of nodes in the GRAPH,

B is the number of branches in the GRAPH, and

O is the storage required for Occupancy tables — typically $m \times 1024$ where $1 \leqq m \leqq 8$ is the number of occupancy tables.

The NEASIM algorithm is designed for rapid execution on the IBM 7090. Average speed depends principally on the size of the GRAPH. Typical speeds range from about 600 trials per second (550-branch GRAPH) to about 5000 trials per second (48-branch GRAPH).

IV. PROBABILITY GENERATOR AND MATCH ROUTINES

The functions of the two routines called PROBABILITY GENERATOR and MATCH were mentioned in the previous section. The present section is concerned with the internal logic of these programs.

The algorithm used by PROBABILITY GENERATOR is due to W. C. Jones.[28] It is felt that this heretofore unpublished algorithm is of sufficiently widespread interest to be included in this paper.

A word on notation: throughout Section IV we shall use the notation $C[A]$ to mean "the contents of A," where A represents some bit in the computer memory. (The reader is therefore cautioned to distinguish between "bit A" and its contents $C[A]$.)

Two computer instructions which will be referred to repeatedly are the logical (inclusive) "OR" and logical "AND" instructions. Instructing the computer to perform an OR on two bits will guarantee that the result will be 1 if, and only if, either or both of the bits contain 1; while an AND yields 1 if, and only if, both bits contain 1. When we OR bit A to bit B, then we shall say that we "perform [A] OR [B];" when we AND bit A to bit B, then we say that we "perform [A] AND [B]."

4.1 *The PROBABILITY GENERATOR*

The purpose of PROBABILITY GENERATOR is to generate the occupancies $p_1, \cdots, p_m$. We mentioned in Section 3.2.2 that this subroutine will fill up $\mathrm{OCT}_{(i)}$ $(i = 1, \cdots, m)$ with 36-bit words each of whose bits contains 1 with probability $p_i (i = 1, \cdots, m)$. In the sequel we will focus our attention on (a representative) one of these 36 bits — keeping in mind that the program is actually working on 36 bits independently and in parallel.

Suppose we wish to generate a random binary variable which takes the value 1 with probability $p (0 < p < 1)$ and to place our result in bit X. The algorithm, when terminated, will give $\Pr\{C[X] = 1\} = p$.

The first action taken by PROBABILITY GENERATOR is to express p as a binary fraction to 10 places.* This fraction is then scanned from right to left until the first bit is found which contains a 1. The algorithm uses this binary fraction of $n \leqq 10$ places determined by the scan. We can therefore, without loss of generality, express p as

$$p = 0.b_n b_{n-1} \cdots b_2 b_1 \qquad b_1 = 1; b_2, \cdots, b_n = 0 \text{ or } 1.$$

A digit selected from a random binary number will be referred to as a "random bit" in the following algorithm. In a random binary number, the value of each digit is 1 with probability $\frac{1}{2}$.

Algorithm:

(i) Set $j = 1$. Generate a random bit, say r_1, and store it in X. Thus $C[X] = r_1$.

* The number of places is arbitrary. Ten was chosen to make the round-off error smaller than 0.001, since occupancies are specified on the Simulation Definition Cards to three decimal places.

(*ii*) If $j = n$, go to step (*vii*). Otherwise, increase j by 1 and continue.

(*iii*) Generate another random bit r_j and store it temporarily in some bit, say R. Thus $C[R] = r_j$.

(*iv*) If $b_j = 0$, go to step (*v*). If $b_j = 1$, go to step (*vi*).

(*v*) Perform [R] AND [X]; store in X. Go to step (*ii*).

(*vi*) Perform [R] OR [X]; store in X. Go to step (*ii*).

(*vii*) Stop.

We observe that step (*ii*) is performed (iterated) exactly n times. Let P_j be the probability that $C[X] = 1$ after the jth iteration. The assertion is that $P_n = p$ — i.e., after the algorithm is terminated, X will contain 1 with probability p.

Proof of Algorithm:

Consider two bits A and B, and let $p_A = \Pr\{C[A] = 1\}$, and $p_B = \Pr\{C[B] = 1\}$. When the contents of A and B are independent, if we perform [A] AND [B], the probability of the result being 1 is

$$p_A p_B ,$$

while if we perform [A] OR [B], the probability of the result being 1 is

$$p_A + p_B - p_A p_B .$$

Now, $\Pr\{r_j = 1\} = \Pr\{r_j = 0\} = \frac{1}{2} \qquad j = 1, \cdots, n.$

Therefore if step (*v*) is executed,

$$P_{j+1} = \tfrac{1}{2} P_j \qquad j = 1, \cdots, n - 1 \tag{3}$$

while if step (*vi*) is executed,

$$\begin{aligned} P_{j+1} &= \tfrac{1}{2} + P_j - \tfrac{1}{2} P_j \qquad j = 1, \cdots, n - 1 \\ &= \tfrac{1}{2} P_j + \tfrac{1}{2}. \end{aligned} \tag{4}$$

But step (*v*) is executed only if $b_{j+1} = 0$, and step (*vi*) is executed only if $b_{j+1} = 1$. Hence (3) and (4) can be combined into

$$P_{j+1} = \tfrac{1}{2} P_j + \tfrac{1}{2} b_{j+1} \qquad j = 1, \cdots, n - 1.$$

Since $P_1 = \frac{1}{2}$, it follows by induction that

$$P_n = \frac{b_n}{2} + \frac{b_{n-1}}{2^2} + \cdots + \frac{b_2}{2^{n-1}} + \frac{b_1}{2^n} = p$$

and our assertion is proved.

4.2 *The MATCH Routine*

The MATCH routine is entered by the program after the busy-idle states have been assigned to the branches. Its purpose is to find whether or not there is a path through the graph for any particular assignment. In the present section we first derive a recursive set-theoretic formula which may be used to determine whether there is a path. The remainder of the section shows how the recursion is carried out by the computer to produce an estimate which converges to the blocking probability of the GRAPH.

In Section 2.2 we saw how the "path enumeration approach" could, in theory at least, be employed in determining the blocking probability. We again take the path enumeration approach, but from a slightly altered point of view:

Instead of trying to find whether there is a path through the *entire* graph at one fell swoop (which amounts to finding whether there is a path from the first node to the last node), we shall try to find whether there is a path from the first node to *each of the nodes* in the GRAPH. While this approach may appear to inject an unnecessary complication, we will see how, by stepping through the GRAPH in an orderly fashion, this approach lends itself naturally to computer programming. We begin by investigating, somewhat further, the geometrical structure of a GRAPH.

Consider the general GRAPH (whose structure is shown schematically in Fig. 7) and suppose that there are a total of ν nodes. Each GRAPH has a first node, N_1, a last node, N_ν, and several intermediate "stages" of nodes. The notion of "stage" is made more precise by the following *definition:* a node N is said to be in stage $s (s = 1, 2, \cdots)$ if, and only if, all paths from N_1 to N contain no more than $s - 1$ branches, and there exists at least one path from N_1 to N which contains exactly $s - 1$ branches.

Any GRAPH will thus contain some number $S \geqq 2$ of stages, where the first and last (Sth stage) consist, respectively, of the single nodes N_1 and N_ν. Let n_s be the number of nodes in stage s, $s = 1, \cdots, S$ (thus $n_1 = n_S = 1$); and define a quantity m_s by

$$m_s = \sum_{i=1}^{s} n_i .$$

We choose to order the nodes of the GRAPH in the following manner: to each of the n_s nodes in stage s ($s = 2, \cdots, S$) assign arbitrarily, but uniquely, one of the integers

$$m_{s-1} + 1, m_{s-1} + 2, \cdots, m_{s-1} + n_s \qquad s = 2, \cdots, S.$$

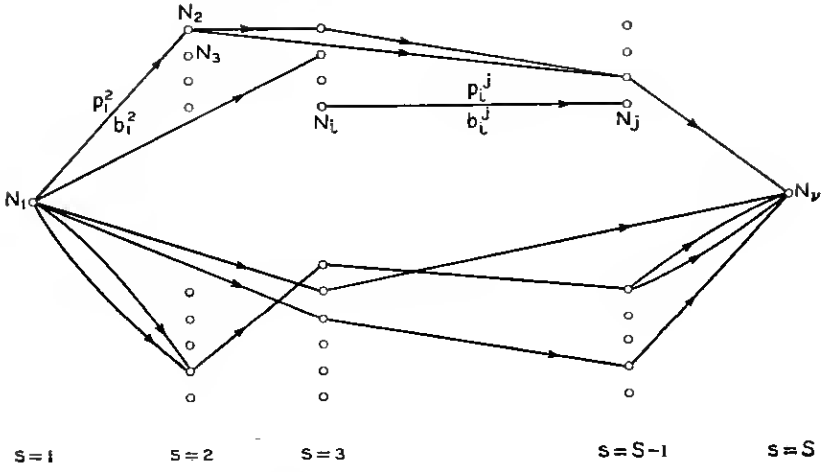

Fig. 7 — A general GRAPH.

The nodes of the GRAPH are hence totally ordered and may therefore be denoted by N_i, where $i = 1, \cdots, \nu$.

When N_i and N_j are connected by one or more parallel branches, where N_i, N_j are arbitrary nodes, we say that, for $i < j$, N_i is *directly connected* to N_j through each of these branches and write $N_i \rightarrow N_j$. We shall suppose, without loss of generality, that if $N_i \rightarrow N_j$, then there is only one connecting branch, b_i^j. (This restriction is assumed in order to avoid introducing yet another index, but will not affect the subsequent results. When a computer instruction involving b_i^j is described, we shall understand that the computer is to execute this instruction for all the b_i^j for which $N_i \rightarrow N_j$.)

Let us now examine the graph, asking the question for every node N_j, "Is there an available path from N_1 to N_j?" Our objective is to answer the question for $j = \nu$.

For each node N_j $(j > 1)$ consider the branches b_i^j for which $N_i \rightarrow N_j$. Clearly, for each of these branches, if there was no available path from N_1 to N_i or there is no available path through the branch b_i^j, or both, then, and only then, will there be no available path from N_1 to N_j which passes through N_i. More precisely, let

X_j be the event: there is no available path from N_1 to N_j,

Y_i^j be the event: there is no available patb from N_1 to N_j which passes througb N_i,

B_i^j be the event: branch b_i^j is not available (busy).

Then

$$Y_i^j = X_i \cup B_i^j$$

$$X_j = \bigcap_{\substack{i<j \\ N_i \to N_j}} Y_i^j = \bigcap_{\substack{i<j \\ N_i \to N_j}} (X_i \cup B_i^j).$$

Consider all the nodes N_i such that $N_i \to N_j$. Suppose there are k_j such nodes. Call them $N_{i_1}, N_{i_2}, \cdots, N_{i_{k_j}}$, where $i_1 < i_2 < \cdots < i_{k_j}$. Next, define an event $Z_j^{(m)}$ recursively by

$$Z_j^{(m)} = (X_{i_m} \cup B_{i_m}{}^j) \cap Z_j^{(m-1)} \qquad m = 2, \cdots, k_j \qquad (5)$$
$$N_{i_m} \to N_j$$
$$j = 2, \cdots, \nu$$

and

$$Z_j^{(1)} = X_{i_1} \cup B_{i_1}{}^j \qquad N_{i_1} \to N_j \qquad (6)$$
$$j = 2, \cdots, \nu.$$

It then follows that

$$Z_j^{(k_j)} = X_j \qquad j = 2, \cdots, \nu \qquad (7)$$

and in particular

$$Z_\nu^{(k_\nu)} = X_\nu$$

where X_ν is the event "there is no path through the graph." The MATCH routine is based upon these formulas.

We now focus our attention on (a representative) one bit in each word of the Node and Branch tables — again keeping in mind that the program is actually working on a full word of bits independently and in parallel. The program will thus execute 36 simultaneous iterations of the MATCH algorithm, which we now proceed to evolve.

When MATCH is entered, the following state of affairs prevails:

(*i*) To each node N_i and to each branch $b_i{}^j$ there has been uniquely assigned one bit of computer memory, so that we can henceforth refer to these *bits* as bit N_i and bit $b_i{}^j$.

(*ii*) $C[b_i{}^j] = 1$ with the appropriate occupancy $p_i{}^j = \Pr\{\text{branch } b_i{}^j \text{ is busy}\}$. (These $p_i{}^j$ were called $p_1, \cdots, p_m$ in Sections II and III.)

(*iii*) The linkage information between the node bits N_i and the branch bits $b_i{}^j$ is stored in the Node-Link and Linkage tables.

It follows that representation of the event $Z_j^{(1)} = X_{i_1} \cup B_{i_1}{}^j$ of formula (6) can be achieved in the computer by performing

$$[N_{i_1}] \text{ OR } [b_{i_1}{}^j]$$

provided that we always set $C[N_1] = 0$; and the event $Z_j^{(k_j)} = X_j$ of (7) can be represented by executing the following steps.

(*i*) Perform $[N_{i_1}]$ OR $[b_{i_1}{}^j]$; store in N_j.
(*ii*) Set $m = 1$.
(*iii*) If $m = k_j$ go to step (*vi*). Otherwise, continue.
(*iv*) Increase m by 1.
(*v*) Perform $([N_{i_m}]$ OR $[b_{i_m}{}^j])$ AND $[N_j]$; store in N_j. Go to step (*iii*).
(*vi*) Stop.

We observe that the algorithm may be condensed if we initially set $C[N_j] = 1 (j \geqq 2)$:

(*i*) Set $m = 1$.
(*ii*) Perform $([N_{i_m}]$ OR $[b_{i_m}{}^j])$ AND $[N_j]$; store in N_j.
(*iii*) If $m = k_j$, go to step (*v*). Otherwise continue.
(*iv*) Increase m by 1. Go to step (*ii*).
(*v*) Stop.

This initialization of the node bits — $C[N_1] = 0$, $C[N_j] = 1$ for $j = 2, 3, \cdots, \nu$ — was mentioned in Section 3.2.3 and can be interpreted as meaning "there is always a path from N_1 to N_1; there is no path from N_1 to N_j $(j > 1)$ until proven otherwise."

Summarizing, the steps that the computer may take to represent the event $Z_\nu^{(k_\nu)} = X_\nu$ are

(M1) Set $C[N_1] = 0$. Set $C[N_j] = 1, j > 1$.
(M2) For each bit N_j, $j = 2, \cdots, \nu$, and in the order

$$N_2, N_3, \cdots, N_\nu$$

find all bits N_i for which $N_i \rightarrow N_j$ and perform

$$([N_i] \text{ OR } [b_i{}^j]) \text{ AND } [N_j]; \text{ store in } N_j.$$

Now, the statement "find all bits N_i for which $N_i \rightarrow N_j$" implies that the input to the program is such that it specifies to the computer which nodes N_i are directly connected to N_j. It was felt that a more straightforward input format would be to specify to which nodes, N_j, each node N_i connects. With the linkage information stored in this fashion,* step (M2) may be replaced by the equivalent step

(M2′) For each bit N_i, $i = 1, \cdots, \nu - 1$, and in the order

$$N_1, N_2, \cdots, N_{\nu-1}$$

* A careful perusal of Section 3.2.2 will show that this is indeed the way the linkage information is stored in the Node-Link and Linkage tables.

find all bits N_j for which $N_i \rightarrow N_j$ and perform

$$([N_i] \text{ OR } [b_i^j]) \text{ AND } [N_j]; \text{ store in } N_j .$$

Steps (M1) and (M2′) constitute the MATCH algorithm for determining whether there is a path through the graph. When the MATCH algorithm is terminated, $C[N_\nu] = 1$ if, and only if, there is no path, so that $\Pr\{C[N_\nu] = 1\} = \Pr\{\text{no path}\}$.

Suppose the algorithm is iterated n times. In any one iteration the event $C[b_i^j] = 1$ is generated with probability p_i^j and independently of the event $C[b_i^j] = 1$ in any other iteration. But $\Pr\{C[N_\nu] = 1\}$ after a given iteration is clearly only a function of the probabilities $\Pr\{C[b_i^j] = 1\}$. Hence, N_ν will contain 1 with the same probability, B, after each iteration, and we conclude that n iterations of the algorithm constitute a sequence of n Bernoulli trials with probability B of success on each trial. Let ξ_i be a random variable such that

$$\begin{aligned} \xi_i &= 1 \text{ if } C[N_\nu] = 1 \text{ after the } i\text{th trial} \\ &= 0 \text{ if } C[N_\nu] = 0 \text{ after the } i\text{th trial} \end{aligned}$$

and let

$$B_n = \xi_1 + \xi_2 + \cdots + \xi_n .$$

An application of the Law of Large Numbers (Ref. 29, p. 189) gives

$$\lim_{n \to \infty} (B_n/n) = B.$$

Identifying B_n/n with $B_G^{(n)}(p_1, \cdots, p_m)$ of Section 3.1 and B with $B_G(p_1, \cdots, p_m)$ of the same section, this last result is equivalent to, and proves the assertion that,

$$\lim_{n \to \infty} B_G^{(n)}(p_1, \cdots, p_m) = B_G(p_1, \cdots, p_m).$$

Repeating the experiment often enough and dividing the number of times $C[N_\nu] = 1$ (i.e., the number of blocked attempts) by the number of iterations n (i.e., the number of attempts) will therefore yield a precise estimate of the blocking probability of the GRAPH. How large n has to be in order to attain any given degree of precision is discussed in the next section.

V. RELIABILITY CONSIDERATIONS

Since any simulation is nothing but an experimental measurement, and hence subject to statistical fluctuations, it is necessary to assess the

reliability of the results. Unfortunately, most of the testing for a given simulation must be done on an *a posteriori* basis and there is, in general, rarely sufficient *a priori* information on which to base the decision of how long the run is to be. The need for such information becomes even more important when the cost factor in computer simulations is considered. It will be shown below that the NEASIM procedure is one which allows for *a priori* determination of run length. The decision of how long the run is to be is based on the desired precision and is made by the computer.

In Section 4.2 we introduced the random variable B_n, the number of times $C[N_\nu] = 1$ in n repetitions of the experiment. B_n will now be called "the number of calls blocked in a simulation run of n calls."* The estimator B_n/n was seen to converge to the blocking probability B, and is indeed a maximum likelihood, mean-unbiased, minimum variance estimator.

To generate the random binary numbers of Section 4.1, NEASIM uses the well-known multiplicative congruential method. This method for pseudo-random number generation has been discussed, tested, and used by numerous investigators[31-36] since it was first proposed by Lehmer.[37] Although the method has been demonstrated to generate 35-bit random binary numbers, there is some measure of cyclic behavior in the low-order bits. NEASIM forms a word of 36 random bits by combining the most random halves (18 high-order bits) of two 35-bit words generated by this method. A further check on the randomness is provided by NEASIM itself, which, as part of its output, prints the generated branch occupancies. We therefore assume that the events $C[N_\nu] = 1$ are independent for individual trials of the experiment. The results of any given computer run should thus be binomially distributed (see also Section 4.2), and hence asymptotically normal.

As an illustration, we again turn to the GRAPH of Fig. 3. For a branch occupancy of $p = p_2 = 0.5$, formula (2) yields $B = 0.0525$. The NEASIM program, for a run of $n = 201{,}600$ calls, gave $B = 0.0529$ — an error within 1 per cent. (This run took some 40 seconds of

* The new nomenclature is chosen to indicate a different interpretation of what NEASIM is doing: NEASIM looks at the configuration of possible paths through the network between two subscribers, takes "snapshots" of the current busy-idle states of the links in these paths, and then finds if there is a path for each snapshot. Thus the program is essentially running calls through a portion of the network — the portion of the network being a representative one, and hence one from which significant statistical data can be extracted to describe the performance of the entire network. A similar approach was taken by A. Feiner, W. C. Jones, and others,[30] who wrote "abbreviated" simulations in which the busy-idle state data were taken from previously run full-scale simulations, but for which a new program had to be written for each new network to be simulated.

computer time.) To study the normality, the number of blocked calls was noted for every 1008 calls run. Since B_n is the number of blocked calls in a run of n calls, the binomial distribution function in this case is

$$\Pr\{B_n \leqq \beta\} = \sum_{k=0}^{\beta} \binom{1008}{k} B^k(1 - B)^{1008-k}$$

and the approximating normal distribution function (Ref. 29, p. 172) is $\Phi(x_{\beta+\frac{1}{2}})$, where

$$x_t = (t - 1008B)h$$

$$h = [1008B(1 - B)]^{-\frac{1}{2}}$$

and

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}y^2} dy$$

is the standard normal distribution function. Fig. 8 is a plot on probability paper of the cumulative frequency distribution determined by the computer, and the theoretical normal distribution line for $B = 0.0525$.

In order to obtain meaningful results, it was felt that the run-length should be determined by the following *criterion:* the number of trials shall be large enough so as to give 95 per cent confidence that the estimator lies within a fixed percentage of the true value of B. Since the blocking probability is unknown at the outset, this criterion is more useful than requiring the estimator to lie in a fixed interval about B. Thus, we wish to choose n large enough so that

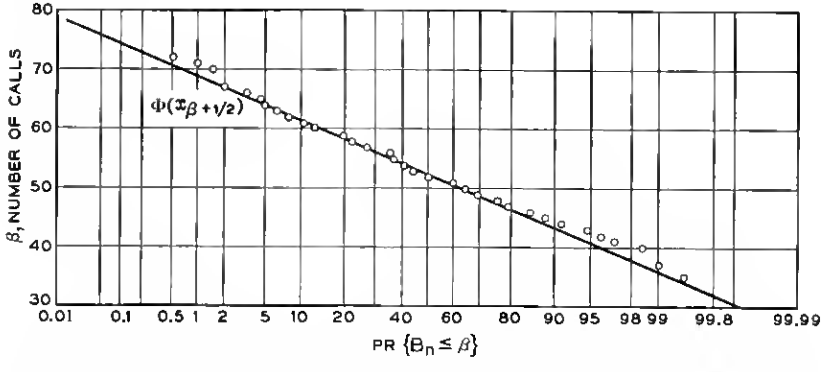

Fig. 8 — Cumulative distribution for GRAPH of Fig. 3 ($p = p_2 = 0.5$) in a run of 201,600 calls in which the number of blocked calls was noted for every 1,008 calls run. The points are experimental data; the line is the theoretical distribution, $\Phi(x_{\beta+\frac{1}{2}})$

$$\Pr\left\{\left|\frac{B_n}{n} - B\right| \leqq aB\right\} \geqq 0.95$$

where a is an input parameter to the program representing the desired percentage. Rearranging terms, we get

$$\Pr\left\{\left|\frac{B_n - nB}{\sqrt{nB(1-B)}}\right| \leqq a\sqrt{\frac{nB}{1-B}}\right\} \geqq 0.95$$

where, for n sufficiently large, the distribution of the random variable

$$\frac{B_n - nB}{\sqrt{nB(1-B)}}$$

approaches the standard normal. Our requirement will therefore be satisfied if

$$a\sqrt{\frac{nB}{1-B}} \geqq 1.96$$

or

$$n \geqq \frac{3.84}{a^2}\left(\frac{1}{B} - 1\right). \tag{8}$$

It follows that the number of trials should be increased as the blocking probability decreases in order to maintain 95 per cent confidence that the estimator lies within a fixed percentage of the true value — as intuition dictates. On the other hand, for a fixed B, as n increases, the 95 per cent confidence limits will narrow. This is displayed in Fig. 9, where the program results for the GRAPH of Fig. 3, for $B = 0.0525$, are seen to lie within the confidence limits.

Furthermore, since $(1/B) - 1 \leqq (1/B)$ for all B in the unit interval, the requirement will be met if

$$n \geqq \frac{3.84}{a^2}\cdot\frac{1}{B}.$$

But for large n, $B \approx (B_n/n)$. Hence, making the substitution we obtain

$$B_n \geqq \frac{3.84}{a^2}.$$

As long as the number of simulated calls is large enough so that the number of blocked calls is at least $3.84/a^2$, there is 95 per cent confidence that B_n/n is within aB of B.

NEASIM was written to accept several values of a ranging from 0.05

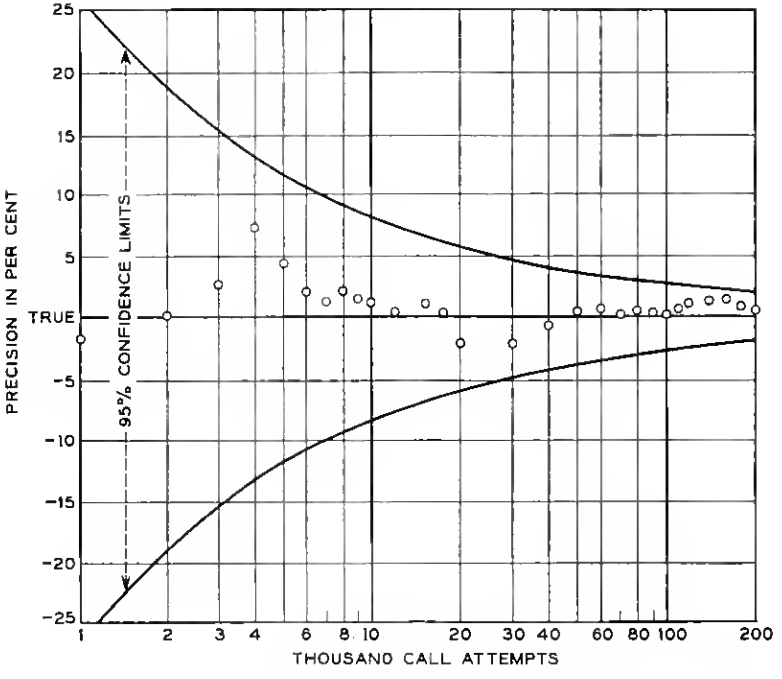

Fig. 9 — NEASIM blocking estimates for GRAPH of Fig. 3 vs run length ($p = p_2 = 0.5$, $B = 0.0525$).

to 1.00. From the specified value, it determines the minimum number of blocked calls necessary to guarantee this precision. Fig. 10 shows the results of the simulation of the GRAPH of Fig. 3 for $a = 0.10$ and 0.50. In GRAPHs where B is very small, it is usually unnecessary to estimate B with a very high degree of precision — especially at the expense of costly computer runs. An upper limit for n is therefore also specified. The computer then proceeds with the simulation until it either exceeds the lower limit on B_n or the upper limit on n.

In the latter case, the reliability can be assessed as follows. Since

$$\Pr\left\{\frac{|B_n - nB|}{\sqrt{nB(1-B)}} \leqq 1.96\right\} \geqq 0.95$$

we can rearrange terms and obtain

$$\Pr\{dB^2 + cB + f \leqq 0\} \geqq 0.95$$

where

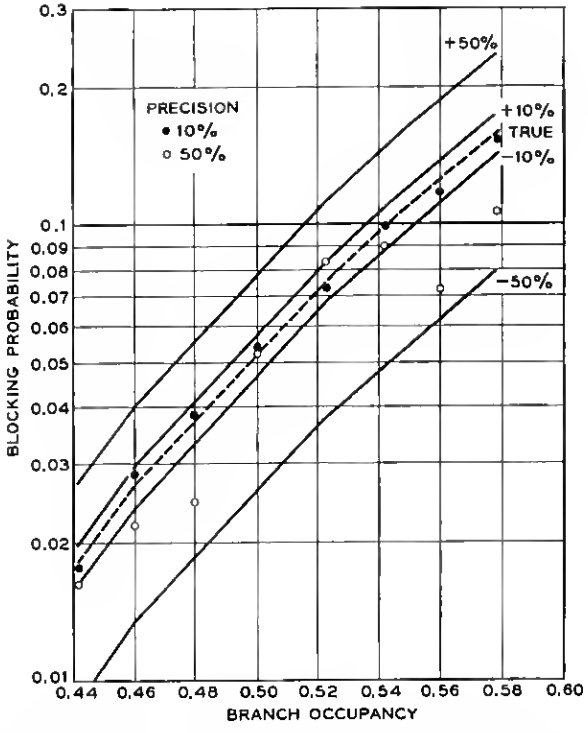

Fig. 10 — NEASIM blocking estimates for GRAPH of Fig. 3 vs branch occupancy $p = p_2$.

$$d = n^2 + 3.8416\, n$$

$$e = -(2nB_n + 3.8416\, n)$$

$$f = B_n^2.$$

It is easily verified that for all n, $B_n > 0$, the parabola $dB^2 + eB + f$ is concave upward, has real roots, and that B_n/n lies between the roots whenever $B_n < n$. Thus for any simulation run, the 95 per cent confidence interval can be obtained by solving for the roots.

Now in all cases of interest, the product of the roots,

$$f/d = B_n^2/(n^2 + 3.8416\, n),$$

can be closely approximated by $(B_n/n)^2$, so that B_n/n can be taken as the geometric mean of the roots. Suppose the higher root is B_1 and the lower root B_2 ; then

$$B_1 = kB_n/n$$

$$B_2 = (1/k)B_n/n.$$

The roots were calculated over a range of values of B_n and n, and the results are displayed in Fig. 11, where k is plotted as a function of B_n/n for various values of n. As an example, suppose that in a run of 80,000 calls, $B_n = 80$ were found blocked. Then $B_n/n = 0.001$ and Fig. 11 gives $k = 1.244$. Hence $B_1 = 1.244\ (0.001) = 0.00124$, $B_2 = 0.001/1.244 = 0.000803$, and there is 95 per cent confidence that $0.000803 \leqq B \leqq 0.001244$. The importance of the case under consideration would then determine whether a longer simulation is necessary.

Since the GRAPH geometry may be as complex as indicated in Fig. 4,

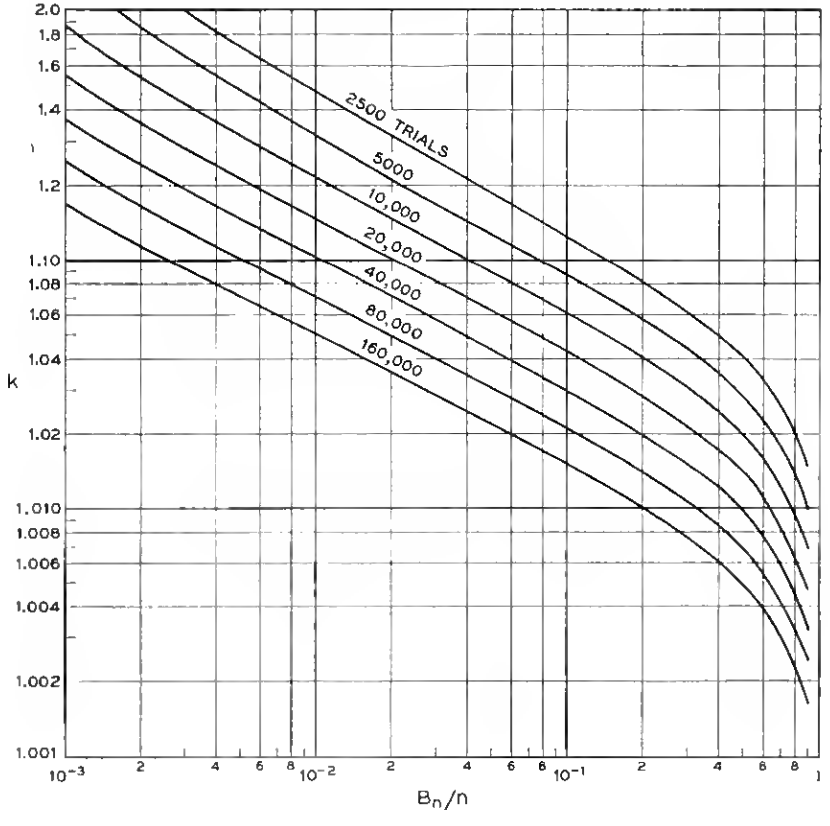

Fig. 11 — The 95 per cent confidence limits for $B: (1/k)(B_n/n) \leqq B \leqq k(B_n/n)$.

it is important to be able to verify that no error was made in mapping the geometry into the computer memory. To this end, a program was prepared by Miss D. Logan which, using the Graph Definition Cards as input and an SC-4020 Microfilm Recorder, draws a picture showing the GRAPH geometry.

VI. POSSIBILITIES FOR INCREASED REALISM

In the preceding section it was shown that the NEASIM program produces precise, reliable results in comparison with Lee's probability linear-graph model of switching networks. Unfortunately, in complex switching networks, substantial differences may exist between the estimates obtained with Lee's analytical technique and actual performance determined by field measurement or full-scale (complete) simulation. These differences appear to be largely attributable to the unreality of the assumption [(e) of Section 2.1] that the link random variables are independent.

The dependence that exists between the different links in a network stage and between the links in adjacent stages is incompletely understood, but nonetheless real. Attempts to take account of interlink dependence by judicious modification of link occupancy values have been moderately successful in calculations and may, of course, be employed in NEASIM runs. However, the nature of the NEASIM process suggests alternate approaches which may ultimately prove to be more fruitful. While considerable success has been obtained with some of the techniques discussed in this section, much remains to be accomplished.

6.1 *Dependence Effects within a Switching Stage*

Two possibilities for increased realism within the confines of a single link stage appear worthy of mention. The NEASIM program typically assigns link stage busy-idle states from a binomial distribution. An obvious suggestion (but one upon which little work has been done) would be to modify the PROBABILITY GENERATOR and/or BUSY-IDLE ASSIGNMENT routines in such a way as to produce busy-idle state assignments taken from various distributions — Jacobaeus' E distribution,[4,6] for example.

A second approach, which has been extensively used, is to incorporate program routines *between* BUSY-IDLE ASSIGNMENT and execution of the MATCH routine. These routines examine the random busy-idle assignments and make appropriate assignment changes where GRAPH

geometry or other considerations indicate. An example of such a routine is one which is designed to insure that there exists at least one "sure-idle" branch out of an $n \times n$ input switch. If calls are only placed between idle network terminals, at least one branch out of an $n \times n$ input switch must be idle. But if the branches leaving the initial GRAPH node are specified at occupancy p, then the program would normally make all these branches busy with probability p^n. The SURE-IDLE routine, upon finding such an assignment, will select an initial branch at random and make it idle. This action, of course, disturbs the otherwise binomial distribution of branch states and should be taken into account when branch occupancy values are specified.

6.2 *Interstage Dependence Effects*

An interesting technique for introducing interstage dependence exists within the NEASIM framework and is based upon an obvious extension of the SURE-IDLE routine just described. Briefly stated, a routine could be designed to examine the busy-idle assignments made on the input and output branches of each node of the GRAPH. Acting on knowledge of the switch geometry and other factors, the "dependencing" routine could change the initial busy-idle assignments where necessary to make them more realistic. Only a very simple and admittedly inaccurate routine has been used to date with, however, a remarkable increase in the "realism" of the results obtained.

The simple-minded DEPENDENCING routine currently employed assumes that every GRAPH node is in reality an $n \times n$ switch. It observes that in an $n \times n$ switch each input branch has probability $1/n$ of being connected to any particular output branch. It attempts to implement its idea of reality by, for each input branch, examining each output branch and forcing the output branch state to agree with the input branch state with probability $1/n$. While this routine can produce rather quaint effects, such as duplicating the state of one input branch on all output branches, it does possess the virtues of simplicity (rapid execution) and a basically correct notion of interstage dependence.

That the use of the SURE-IDLE and DEPENDENCING routines can be effective is demonstrated in Fig. 12. The results shown in the figure are for a moderately complex eight-stage switching network whose GRAPH has 232 branches. NEASIM results with and without dependencing and sure-idles are compared with the results of a full-scale simulation. The improvement in realism possible with the rudimentary routines just described appears quite dramatic.

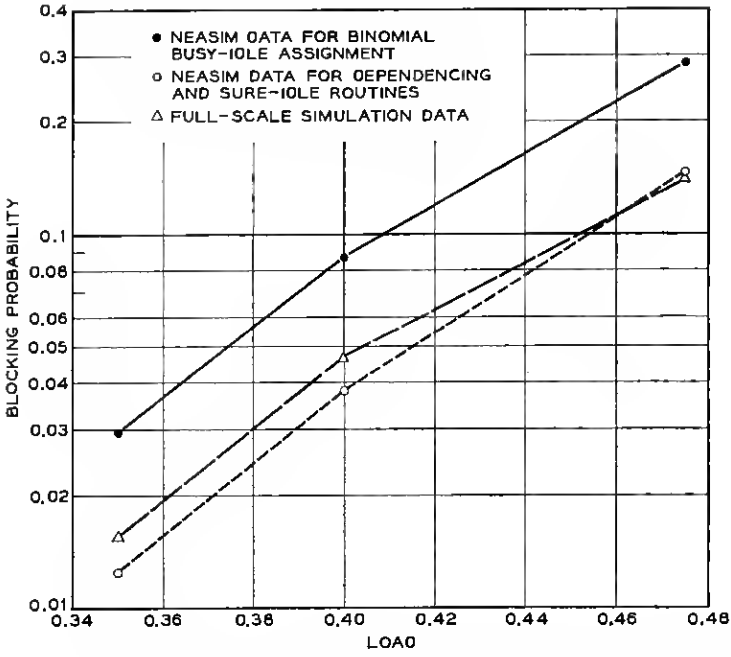

Fig. 12 — Comparison of NEASIM results (with and without dependencing and sure-idles) with full-scale simulation data for a realistically complex network.

VII. CONCLUSION

The NEASIM approach to switching network simulation has achieved its major goal, the mechanization of a widely employed — if somewhat unrealistic — technique for load-loss analysis. The applications of the GRAPH model need no longer be restricted by the overpowering computational difficulties brought on by GRAPH complexity.

Furthermore, the simulation of an analytical model concept basic to NEASIM seems to open new areas for profitable exploration in the analysis of switching systems — and perhaps other stochastic systems as well. The success of the elementary realism-injecting routines suggests that further research along this line may be rewarding.

Finally, the degree of realism in results attained so far, coupled with the ease of application, has produced what amounts to a new tool for use in both the design and engineering of new switching networks. It is now feasible to achieve relatively complete and accurate load-loss engineering data on complex switching systems well in advance of actual field experience.

VIII. ACKNOWLEDGMENTS

The authors wish to express their appreciation to W. S. Hayward, A. Descloux and J. G. Kappel for many stimulating discussions. In particular, Mr. Hayward suggested a significant improvement in the MATCH routine. The work of Miss D. Logan in writing several parts of the program is greatly appreciated.

REFERENCES

1. Kosten, L., The Historical Development of the Theory of Probability in Telephone Traffic Engineering in Europe, Teleteknik, **1**, 1957, pp. 32–40.
2. Wilkinson, R. I., The Beginning of Switching Theory in the United States, Teleteknik (English Edition), **1**, 1957, pp. 14–31.
3. Syski, R., *Introduction to Congestion Theory in Telephone Systems*, Oliver and Boyd, London, 1960.
4. Jacobaeus, C., Blocking Computations in Link Systems, Ericssou Review, No. 3, 1947, pp. 86–100.
5. Lundkvist, K., Method of Computing the Grade of Service in a Selection Stage Composed of Primary and Secondary Switches, Ericsson Review, No. 1, 1948, pp. 11–17.
6. Jacobaeus, C., A Study on Congestion in Link Systems, Ericssou Technics, **48**, 1950, pp. 1–68.
7. Jensen, A., A Basis for the Calculation of Congestion in Crossbar Systems, Teleteknik, No. 3, 1952.
8. Lee, C. Y., Analysis of Switching Networks, B.S.T.J., **34**, 1955, pp. 1287–1315.
9. Elldin, A., Applications of Equations of State in the Theory of Telephone Traffic, Thesis, Stockholm, 1957.
10. Fortet, R., aud Canceill, B., Probabilité de Perte en Selection Conjuguee, Teleteknik, **1**, 1957, pp. 41–55.
11. LeGall, P., Methode de Calcul de L'encombrement dans les Systèmes Téléphoniques Automatiques a Marquage, Ann. des Telecom., **12**, 1957, pp. 374–386.
12. Beneš, V. E., Heuristic Remarks and Mathematical Problems Regardiug the Theory of Connecting Systems, B.S.T.J., **41**, 1962, pp. 1201–1247.
13. Beneš, V. E., Algebraic and Topological Properties of Connecting Networks, B.S.T.J., **41**, 1962, pp. 1249–1274.
14. Beneš, V. E., A "Thermodynamic" Theory of Traffic in Connecting Networks, B.S.T.J., **42**, 1963, pp. 567–607.
15. Beneš, V. E., Markov Processes Representing Traffic in Counecting Networks, B.S.T.J., **42**, 1963, pp. 2795–2837.
16. Frost, G. R., Keister, W., and Ritchie, A. E., A Throwdown Machine for Telephone Traffic Studies, B.S.T.J., **32**, 1953, pp. 292–359.
17. Gerlough, D. L., Simulation of Freeway Traffic by an Electronic Computer, Proc. Highway Research Board, 1956, pp. 543–547.
18. Goode, H. H., Pollmar, C. H., and Wright, J. B., The Use of a Digital Computer to Model a Signalized Intersection, Proc. Highway Research Board, 1956, pp. 548–557.
19. Clapham, J. C. R., A Monte Carlo Problem in Underground Communications, Operations Research Quarterly, **9**, No. 1, 1958, pp. 36–54.
20. Jennings, N. H., and Dickens, J. H., Computer Simulation of Peak Hour Operation in a Bus Terminal, Management Science, **5**, No. 1, 1958, pp. 106–120.
21. Stark, M. C., Computer Simulation of Street Traffic, NBS Tech. Note 119, U.S. Dept. of Commerce, Off. Tech. Services, Washington, D.C., 1958.
22. Gross, F. J., Simulation of Data Switching Systems on a Digital Computer, A.I.E.E. Transactions, Part I — Communications and Electronics, **46**, 1960, pp. 796–800.

23. Katz, J. H., Simulation of a Traffic Network, Communications of the ACM, **6**, 1963, pp. 480–486.
24. Dietmeyer, D. L., Gordon, G., Runyon, J. P., and Tague, B. A., An Interpretive Simulation Program for Estimating Occupancy and Delay in Traffic-Handling Systems Which Are Incompletely Detailed, A.I.E.E. Conference Paper CP 60-1090, San Diego, 1960.
25. Gordon, G., General Purpose Systems Simulator, Proc. Eastern Joint Computer Conference, 1961, pp. 87–104.
26. Weber, J. H., Some Traffic Characteristics of Communications Networks with Automatic Alternate Routing, B.S.T.J., **41**, 1962, pp. 769–796.
27. Bader, J. A., and Hayward, W. S., Computer Simulation as a Machine Aid to Switching System Design, A.I.E.E. Conference Paper CP 61-258, New York, 1961.
28. Jones, W. C., personal communication.
29. Feller, W., *An Introduction to Probability Theory and Its Applications*, John Wiley and Sons, New York, 1957.
30. Feiner, A., Jones, W. C., *et al.*, personal communication.
31. Juncosa, M. L., Random Number Generation on the BRL High-Speed Computing Machines, Report No. 855, Ballistics Research Laboratories, Aberdeen Proving Ground, Maryland, 1953.
32. Todd, J., and Taussky, O., Generation of Pseudo-Random Numbers, *Symposium on Monte Carlo Methods*, ed. Meyer, H. A., John Wiley and Sons, New York, 1956, pp. 15–28.
33. Bofinger, E., and Bofinger, V. I., A Periodic Property of Pseudo-Random Sequences, Jour. ACM, **5**, 1958, pp. 261–265.
34. Certaine, J. E., On Sequences of Pseudo-Random Numbers of Maximal Length, Jour. ACM, **5**, 1958, p. 353.
35. Greenberger, M., Appendix to Part II of Decision-Unit Models and Simulation of the United States Economy, M.I.T., January, 1958.
36. Greenberger, M., Random Number Generators, Preprints of the 14th Natl. Conf. ACM, September, 1959.
37. Lehmer, D. H., Mathematical Methods in Large-Scale Computing Units, *Proc. of a Second Symposium on Large-Scale Digital Calculating Machinery*, Ann. of the Computation Laboratory of Harvard University, **26**, 1951, p. 141.